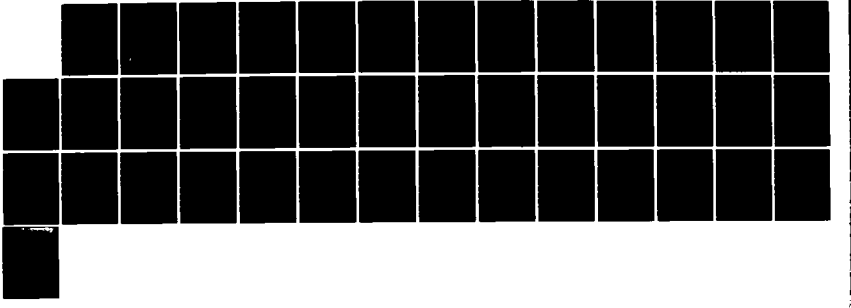
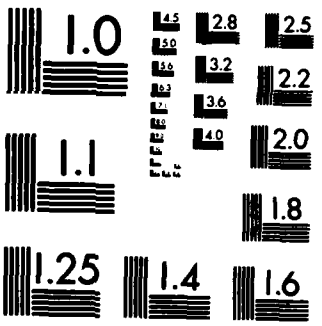
AD-A146 577 ARCHITECTURE DESIGN AND SYSTEM; PERFORMANCE ASSESSMENT 1/1
AND DEVELOPMENT ME.. (U) NAVAL SURFACE WEAPONS CENTER
SILVER SPRING MD J FRANKLIN ET AL. 30 DEC 83
UNCLASSIFIED NSWC/TR-83-324-VOL-1 F/G 9/2 NL





COPY RESOLUTION TEST CHART

(12)

NSWC TR 83-324

ARCHITECTURE, DESIGN, AND SYSTEM; PERFORMANCE ASSESSMENT AND DEVELOPMENT METHODOLOGY FOR COMPUTER-BASED SYSTEMS: VOL. I–METHODOLOGY DESCRIPTION, DISCUSSION, AND ASSESSMENT

BY J. FRANKLIN, C. GRAY, JR.,
ARTHUR WRENN

UNDERWATER SYSTEMS DEPARTMENT

30 DECEMBER 1983

NAVAL SURFACE WEAPONS CENTER

Dahlgren, Virginia 22448 • Silver Spring, Maryland 20910

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NSWC TR 83-324	2. GOVT ACCESSION NO. AD-A 146 577	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ARCHITECTURE, DESIGN, AND SYSTEM; PERFORMANCE ASSESSMENT AND DEVELOPMENT METHODOLOGY FOR COMPUTER-BASED SYSTEMS: VOL. I - METHODOLOGY DESCRIPTION, DISCUSSION, AND ASSESSMENT		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Franklin, C. Gray, Jr., and Arthur Wrenn		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center (Code U32) White Oak Silver Spring, MD 20910		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work request: N00C 2482 WR-10462 task # 24444
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 30 December 1983 13. NUMBER OF PAGES 41
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Combat System Engineering
Computer System Modeling
System Engineering
Computer System Assessment

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Volume I of this report describes, discusses, and assesses a performance assessment and development methodology for computer-based systems comprised of MSI (medium scale integration) and LSI (large scale integration) computers. The report is not meant to be a user's manual or a detailed documentation description of the methodology. In addition to describing the methodology, one chapter is devoted to a discussion of testing versus analytic assessment.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Cont.)

Volume II of the report develops in detail some of the validity data used in Chapter 4 of Volume I by applying the methodology to the Mk 116 Mod 1 and Mod 0 Fire Control Systems.

FOREWORD

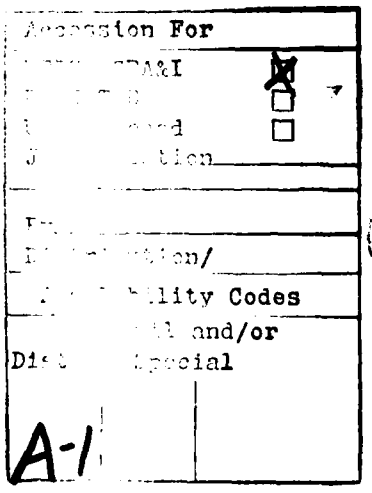
Volume I of this report describes, discusses, and assesses a performance assessment and development methodology for computer-based systems comprised of MSI (medium scale integration) and LSI (large scale integration) computers. The report is not meant to be a user's manual or a detailed documentation of the methodology's software tools. In addition to describing the methodology, one chapter is devoted to a discussion of testing versus analytic assessment.

Volume II of the report develops in detail some of the validity data used in Chapter 4 of Volume I by applying the methodology to the Mk 116 Mod 1 and Mod 0 Fire Control Systems. In addition Volume II illustrates how the methodology is applied to a system.

The authors wish to acknowledge Phillip Hwang for rigorously reviewing Volume I of the report.

Approved by:

J. E. GOELLER, Head
System Engineering Division



CONTENTS

Chapter		Page
1	INTRODUCTION AND REPORT OVERVIEW	1
2	THE METHODOLOGY	5
3	THE AUTOMATED ASSESSMENT TOOLS	15
4	VALIDITY AND APPLICATIONS	27
5	CONCLUSIONS	29
6	COMPUTER SYSTEM ASSESSMENT VIA THE METHODOLOGY VERSUS DIRECT MEASUREMENT	33
Appendix		
A	SOFTWARE FORMAT STANDARDS FOR MAXIMIZING THE AUTOMATED CAPABILITIES OF THE CSE TOOLS	A-1

ILLUSTRATIONS

Figure		Page
1	AUTOMATED ARCHITECTURE TOPOLOGY PROGRAM OUTPUT	6
2	EXAMPLE OF DEVELOPMENT OF THREAD EQUATION AND QUEUEING NETWORK (STEPS 1 THROUGH 8 OF THE METHODOLOGY).	9
3	STEP 4 OF THE METHODOLOGY: END IN DEVELOPMENT TECHNIQUE . . .	11
4	THE METHODOLOGY'S STEPS AND TOOLS	22
5	EXAMPLE OF AN ARCHITECTURE THAT THE TIMING PROGRAM MAY GIVE INVALID RESULTS	25

TABLES

Table		Page
1	AUTOMATED ARCHITECTURE TABLE PROGRAM OUTPUT	7
2	THE METHODOLOGY'S MEASURES OF EFFECTIVENESS	13
3	OUTPUT OF THE DATA EXTRACTION (DATA BASE BUILD) PROGRAM	16
4	OUTPUT SOFTWARE TIMING PROGRAM	18
5	OUTPUT OF SOFTWARE TIMING PROGRAM	21

CHAPTER 1

INTRODUCTION AND REPORT OVERVIEW

Current development methodology as it is implemented for computer-based systems is:

1. Formulate a set of requirements;

These requirements usually fall into three categories: (a) data requirements, requirements that specify the inputs and ouputs; (b) performance requirements, requirements that specify rates and acceptable delays of inputs and outputs; and (c) operational requirements, requirements that specify the operational environment.

2. Develop a set of functions that when performed produce outputs that meet the "data requirements".

3. Iteratively partition these functions into more detailed subfunctions, combine into common subfunctions, and allocate functions/ subfunctions to servers. (A server is anything that performs functions, e.g., hardware, software, or people. Because of the input/output nature of the "data requirements" functions will have an interconnectivity. Since functions are assigned to servers, this may necessitate the servers also having an interconnectivity.)

The reason common function/subfunctions are combined is because of the implicit assumption that the number of servers available is not an unlimited resource or the servers/interconnectivity are already specified. With a performance driven methodology (that will be described), functional combination should not be done until after the performance indicates the servers in the first cut/natural allocation are overloaded. If overload is indicated than this natural allocation will need to be altered by functional combination, reallocation, or altering the server/function interconnectivity to form a more efficient allocation. (This more efficient allocation usually does not map one-to-one with the natural functional flow of the system.)

This function/subfunction allocation to the system of servers is commonly referred to as an architecture. As the iterations in 3 become more and more detailed, the functional allocation is referred to as a "design."* The process

*One point of view is, that the point of transition from an architecture to a design occurs when the functions are allocated to servers to reflect the way the system will actually operate. In practice, however, it is usually some arbitrary level of detail defined by the systems engineer.

of translating a design into software is actually just an extension of the process of finer partitioning of the subfunctions, i.e., software instructions being viewed as very low level subfunctions. At the software phase of the development process, these software instructions (subfunctions) are allocated to servers like CPU'S, IO channels, disks, memory, busses, etc. There is no theoretical reason for limiting the functional allocation/architecture to any particular level and in point of fact this allocation process could be extended to the detail of the molecular or atomic level. In practice, however, the process stops at a level where the allocation can be easily worked with, i.e., a level that will produce the desired system with minumum cost and effort.

The current development methodologies (steps 1 through 3), if scrupulously implemented, by their very nature insure that the "data requirements" (as defined in step 1), are met. The reason being that the current methodologies specify how the data requirements are to be developed. However, there is nothing in the methodologies that insures that "performance requirements" are met, because the current methodologies do not specify how to assess performance. Some system engineering/software engineering methodologies have added an ad hoc performance assessment step, but usually do not provide the details of how it is to be implemented.

Numerous proprietary tools exist to assess performance. These tools either have a queue theoretic or event stepped simulation basis. However, these tools do not provide techniques for generating their necessary inputs, (e.g., service/ execution times, service frequencies, branching probabilities, funtional flows, etc.).

Without a performance assesment methodology it is not possible to:

1. assess if the architecture/design meets the performance requirements,

2. evaluate alternative architecture/designs,

3. perform quantitative architecture/design trade off studies.

It is the purpose of this report to present a complete and useable performance driven development methodology that is applicable to each iteration of the functional allocation process. From the discussion given above on functional allocation this means the methodology is applicable throughout the entire development process.*

Chapter 2 presents an overview of the methodology. Chapter 3 discusses the software tools of the methodology, discusses how the analyst utilizes the tools' outputs to implement the methodology, and discusses a limitation of the methodology as applied to a specific architecture type. Chapter 4 discusses the

*If "maintenance" is viewed as just another iterative application of the development process, then this performance driven methodology is applicable over the entire life cycle of the system, which indeed it is. If the methodology is applied at the points in the life cycle where developement has completed and the product is being delivered then the methodology acts as a product assurance methodology.

validity of the methodology in terms of percent error and also lists the systems to which it has been applied. Chapter 5 presents conclusions on the methodology's validity, applications, and optimal application environment. Chapter 6 is a discussion of assessing a system by using the methodology or by measuring the system with either a hardware or software monitor. Appendix A presents a scheme for the optimal employment of the methodology by formatting the software to allow for machine readable embedded comments. Volume II develops some of the validity data used in Chapter 4 by applying the methodology to the Mk 116 Mod 1 and Mod 0 Fire Control Systems.

CHAPTER 2

THE METHODOLOGY

The purpose of this chapter is to present a development methodology that contains both data and performance methodology elements. Details of the data/function generation elements are kept to a minimum because they are well defined in existing development methodologies as discussed in Chapter 1. The methodology to be presented will fill the need for a performance driven development methodology.

Although this performance methodology is applicable to both systems that are under development and existing systems that have been completed, what is to be presented is written with the point of view of developing new systems.

The steps of the methodology are:

1. Using the latest development information, create a list of functions and function frequencies (i.e., number of function executions per time interval) that the development system will perform.

2. Find existing (software coded) systems that have high level functions identical to or similar to the development system.

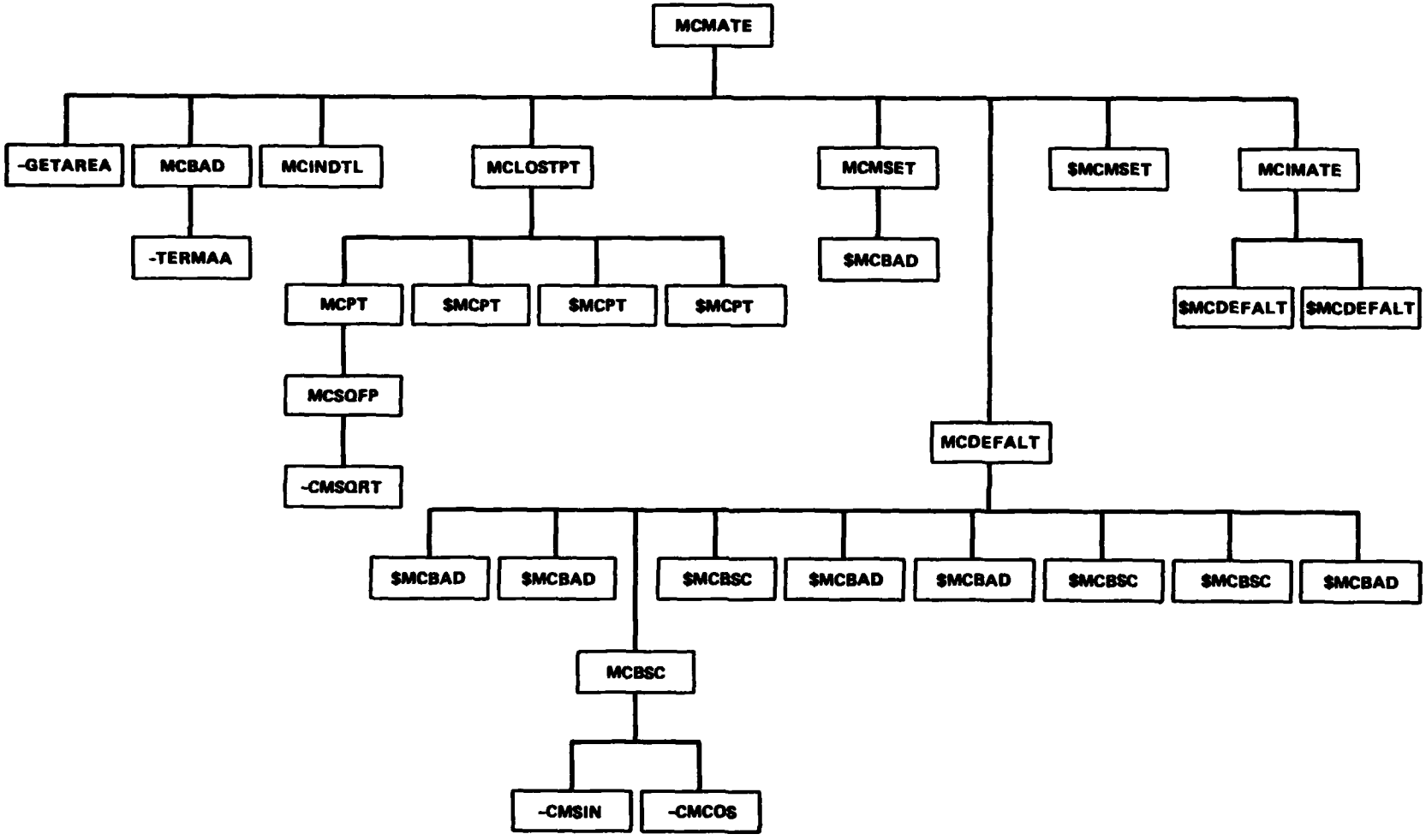
This is not as hard as it sounds because new systems are usually evolutions of existing systems or incorporations of successful approaches from implemented/ prototype systems or algorithms.

3. Using an automated tool (see Chapter 3) applied to the existing systems' software, generate the detailed architecture of the existing system (that is of interest as defined by step 1). See Figure 1 and Table 1 for an example of the output of the automated tool.

The "description" column of Table 1 is supplied by the analyst by consulting the existing system's Program Design Spec (PDS) or the comments in the code. The "frequency" and "execution time" columns are filled in steps 5 and 6.

(It should be noted that step 3, step 5, and step 6 document the architecture/ design of existing computer systems. Usually the detailed software architectures of existing systems never get documentated. This generated documentation should be useful to analysts, system engineers, and maintenance and design personnel.)

4. Map the functions of the existing system onto the architecture of the development system.



$ = ADDITIONAL LOWER LEVEL PROCEDURES CALLED BUT NOT DRAWN

FIGURE 1. AUTOMATED ARCHITECTURE TOPOLOGY PROGRAM OUTPUT

TABLE 1. AUTOMATED ARCHITECTURE TABLE PROGRAM OUTPUT

PROCEDURE NAME	PROCEDURE DESCRIPTION	PROCEDURE FREQUENCY	PROCEDURE EXECUTION TIME
1. MCMATE			
1. 1. -GETAREA			
1. 2. MCIMATE			
1. 2. 1. MCDEFALT			
1. 2. 1. 1. MCBSC			
1. 2. 1. 1. 1. -CMSIN			
1. 2. 1. 1. 2. -CMCOS			
1. 2. 1. 2. MCBSC	SEE ALL PROCEDURES BEGINNING WITH 1. 2. 1. 1.		
1. 2. 1. 3. MCBSC	SEE ALL PROCEDURES BEGINNING WITH 1. 2. 1. 1.		
1. 2. 1. 4. MCBSC	SEE ALL PROCEDURES BEGINNING WITH 1. 2. 1. 1.		
1. 2. 1. 5. MCBAD			
1. 2. 1. 5. 1. -TERMAA			
1. 2. 2. MCDEFALT	SEE ALL PROCEDURES BEGINNING WITH 1. 2. 1.		
1. 2. 3. -CMBAMS			
1. 2. 4. MCBSC	SEE ALL PROCEDURES BEGINNING WITH 1. 2. 1. 1.		

The outputs of step 3 (e.g., Table 1 and Figure 1) detail all the functions and their subfunctions* of the existing system. These functions and subfunctions are organized in a hierarchy. The single digit number is assigned to the highest function and the subfunctions are assigned multidigit numbers. Thus 1 calls 1.1, 1.2, 1.3, etc.; and 1.1 calls 1.1.1, 1.1.2, etc.; and 1.1.1 calls 1.1.1.1, 1.1.1.2, etc. (See the first line of Figure 2.) This hierarchy can be expressed as enumerated threads of functions/procedures, e.g., 1 + 1.1 + 1.1.1, 1 + 1.2 + 1.2.1, etc. If desired the individual threads that make up the hierarchy can be summed to form one large thread for the entire hierarchy.

The mappping of functions of the existing system (picked in step 2) onto the architecture of the development system is a process of deciding which existing system threads or even what individual subfunctions within a thread should be assigned to a function of the development system. Thus every function of the development system will be modeled by a combination of existing system threads/thread pieces which in itself will form a thread, and will be called a development system function thread or a model thread. In short, the model threads are a reduction of a system architecture (hierarchy) into what the analyst deems is the needed information/functions necessary to characterize the system for which the desired model is to be built.

(If this methodology is being applied for the purpose of modeling an existing system rather than for the purpose of modeling a development system, then step 4 still must be done except the assigning of the thread to a development system function is omitted.)

Usually all subfunctions of the existing system are included in the function to be modeled in the development system even if some of the existing system's subfunctions are not specifically called out by the development system's current documentation. The rationale for this is that the development system has not progressed far enough to consider this level of detail, and when it does, it will have to somehow include these subfunctions. If on the other hand, certain subfunctions of the existing system perform activities that will be assigned to a separate function of the development system, then these subfunctions will be excluded from the model of the development system's thread. Examples of including and excluding subfunctions from a thread are:

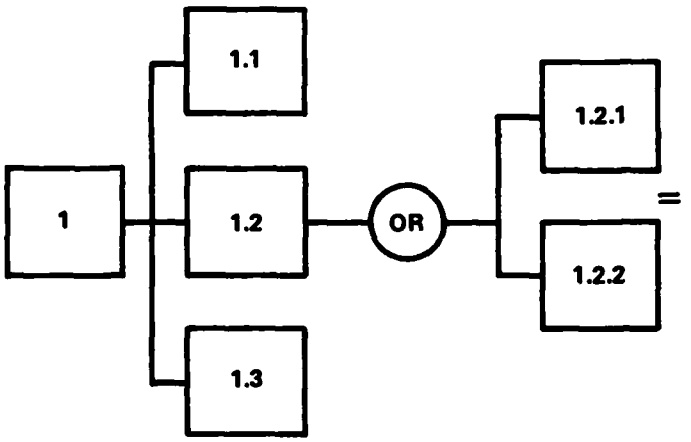
a. The existing system subfunctions include sin, cos, or square root procedure calls, whereas the development system does not talk to this level of detail. Thus sin, cos, etc. subfunctions will be included in the development model thread.

b. The existing system subfunctions include disk Input/Output (I/O) and the development system will have similar disk resident files even though the I/O has not been designed yet. Thus these I/O subfunctions would be included in the development model thread.**

*In Figure 1 the term (software) procedure has been substituted for the term function. As was indicated in Chapter 1, the names of items may change at different phases of the development process, but the actual development process itself is just a repetitive iteration of the same function/allocation process. Thus at the software phase of development we are allocating procedures. Procedures called by parent procedures are equivalent to subfunctions.

**Swapping and Roll in/Roll out would be included in this rationale.

FUNCTIONAL OR SOFTWARE ARCHITECTURE



ARCHITECTURE TABLE

PROCEDURES IN THREAD	PROCEDURE TIME	FREQ.
1	2 + 4*A	6
1.1	8	1
1.2	5	A
1.2.1	8*B + C*30	1
1.2.2	10*D	0
1.3	7*20	0.8

THREAD TIME EQUATION $= 6\left[2 + 4*A + 8*1 + 5*A + A\left(1\left\{8*B + 30*C\right\} + 0\ \left\{10*D\right\}\right) + 7*20*.8\right]$

QUEUEING NETWORK

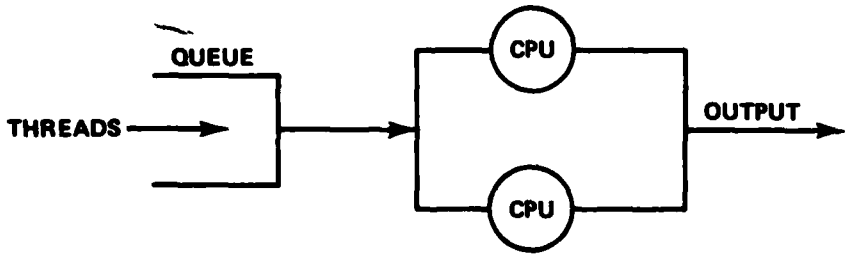


FIGURE 2. EXAMPLE OF DEVELOPMENT OF THREAD EQUATION AND QUEUEING NETWORK (STEPS 1 THROUGH 8 OF THE METHODOLOGY)

c. The existing system subfunctions formulate and send messages, whereas the development system has a separate message formulating/sending function. Thus all message subfunctions of the existing system would be excluded from the thread.

With this scheme, the usual objection to modeling a system in development; namely, that the development of the system has not progressed far enough to know its detailed subfunctions, can be overcome by approximating the development system with low level detailed subfunctions from the existing system. As the development system information becomes more detailed, it replaces information used from the existing system, i.e., the assessment modeling becomes more accurate. (The method of design that utilizes top level functions from the development system and bottom level knowledge/functions from existing systems can be characterized as an "End in Design Technique." See Figure 3.)

5. The execution times of the subfunctions/procedures that make up the model thread are developed by applying an automated timing tool to the existing system's (or if available the development system's) code. Details of this timing tool and its use will be given in Chapter 3.

6. The frequency of occurrence of each subfunction/procedure in a thread is either the same as the root function* of the thread (the frequency of the root function was developed in Step 1); or a multiple of the root function because the function is in a loop; or a fraction/probability of the root function because the subfunction/procedure is conditionally called in an IF Test/branch. The automated timing tool indicates whether a subfunction/procedure is called from a loop or a branch and the address of each. However, the analyst must look up in the code being analyzed the actual variable/value controlling the loop or condition of the branch. These loop and branch variables will become the parameters of sensitivity of the architecture/performance model.**

7. By combining the information developed in steps 4, 5, and 6 the execution time of the thread (e.g., 1 + 1.1, etc.) can now be expressed as a thread time equation. See the first and second lines of Figure 2. In Figure 2 proc 1 is made up of a segment 2 time units plus a loop that is 4 time units and looped "A" number of times. Proc 1.2 is called within the "A" loop and thus its frequency of call is "A." Since 1.2.1 and 1.2.2 are mutually exclusive due to the "or," the frequency of 1.2.2 is zero. Proc 1.3 is conditionally called with frequency .8. The remainder of Figure 2's architecture table is interpreted in a similar manner. Note that in developing the thread time equation the frequencies of the individual called procedures cascade back toward the root function. (See the thread time equation of Figure 2.) The information contained

*Root Function = function labeled with a single digit number in architecture table (Table 1). The root function frequency is the thread's frequency.

**The quantitative values of these parameters of sensitivity may have to be supplied from the operational scenario or from the characteristics of the elements of the combat system. These parameters of sensitivity are the links that connect the architecture model with the rest of the combat system (or a systems analysis model of the combat system) or inputs from the real world.

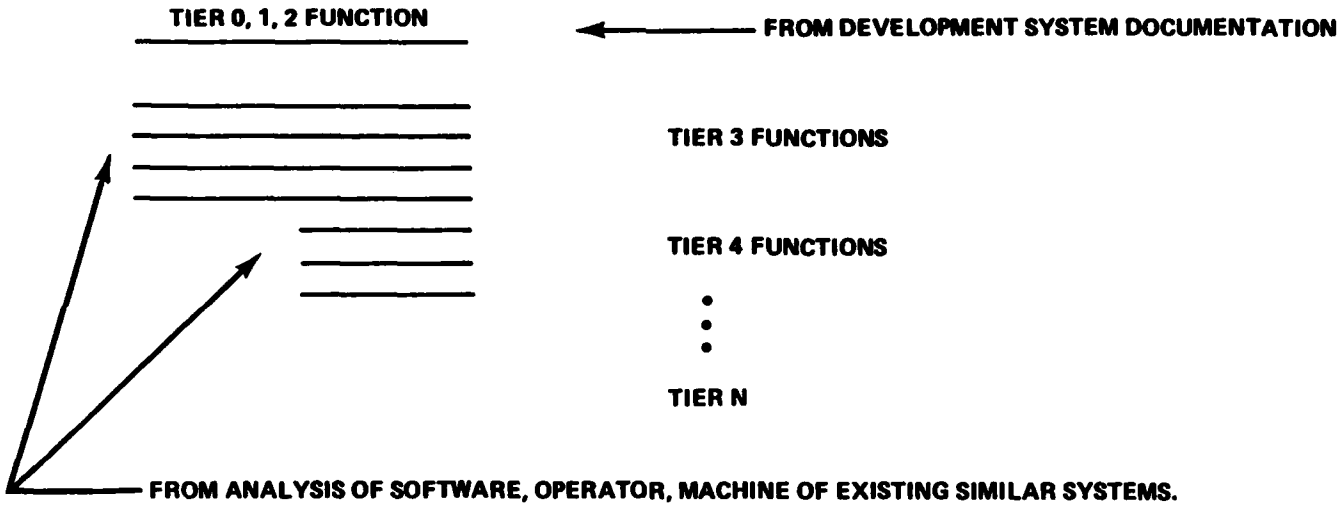


FIGURE 3. STEP 4 OF THE METHODOLOGY : END IN DEVELOPMENT TECHNIQUE

in Figure 2's architecture table (and thread time equation) is typical of the type of information that can be extracted from existing software using the tools of the methodology (see Chapter 3).

8. Represent the servers (computer hardware, human operators, etc.) of the development system as a queueing network. See the third line in Figure 2.

9. Express the threads, threads' frequency, thread time equations, and the queueing network mathematically in terms of a set of queueing theory equations.

10. Solve the queueing equations for queueing measures of effectiveness such as given in Table 2. These MOEs give the performance assessment of an architecture.

Steps 9 and 10 have been automated for most of the standard computer architectures. The tool used at NSWC is POD. See Reference 1.

11. If Steps 8 and 9 cannot be accomplished analytically, solve the queueing network with an event stepped/state space simulation.

12. As the development proceeds iterate Steps 1 through 10 with updated/ current information.

TABLE 2. THE METHODOLOGY'S MEASURES OF EFFECTIVENESS

- AVERAGE RESPONSE TIME
- AVERAGE WAIT TIME
- AVERAGE DELAY
- QUEUE LENGTH
- CONTENTION
- BOTTLENECKS
- PROBABILITY WAIT TIME EXCEEDS A GIVEN VALUE
- PROBABILITY QUEUE LENGTH EXCEEDS A GIVEN VALUE
- PROBABILITY RESPONSE TIME EXCEEDS A GIVEN VALUE
- THROUGHPUT VERSUS LOAD

CHAPTER 3

THE AUTOMATED ASSESSMENT TOOLS

DESCRIPTION

The automated assessment tools consist of three computer programs.* These programs are run in sequence. The first program (the FORMATTING PROGRAM) merely reads the (tactical, etc.) software (that is to be timed) that has been generated on its native machine** and formats it into fixed record, fixed block, ASCII that can be read efficiently by a CDC CYBER 720. The second program (the DATA EXTRACTION PROGRAM) scans this formatted software's assembly/machine level instructions to extract jumps and procedure calls on a per procedure basis. An example of the output of this second program is given in Table 3. The output of the Data Extraction Program is actually a data base of the procedure's machine level jump instructions, the sums of the instruction times between the jumps, and the procedure's calls to units outside itself.† A path within a procedure can be characterized for timing purposes by a specific set of jumps. Since the time along each jump segment of a path is known (i.e., the sums in the data base), a path time can be calculated by summing the jump segment times. This is

*The programs are executed on a Control Data Corp (CDC) CYBER 170/720, are written in COBAL, and use a CDC utility (Record Manager) to build and query an index sequential data base. The timing programs currently will time code written in AN/UYK-20/CMS 2M, AN/UYK-7/CMS 2Y, Digital Equipment Corp. (DEC) PDP 11/FORTRAN IV, and also assembly langugage for these machines.

**The input to the Formatting Program is a specific compiler output option (generated on the native machine) that lists a line of the high order language (HOL) and the corresponding assembly instructions the HOL line compiles to. This compiler output option is sometimes called a side-by-side listing.

†The Data EXTRACTION program contains a table of all the existing system's instruction types and their execution times. If the existing system and development system have different computer types (e.g., AN/UYK-20 versus AN/UYK-7) then the execution times listed in this table can be modified to reflect what they would be on the development system's computer. Utilization of the table in this manner would allow a model to be built of a futuristic computer that had not yet been built.

TABLE 3. OUTPUT OF THE DATA EXTRACTION (DATA BASE BUILD) PROGRAM

PROC NAME	SYS PROC OF PROC	INSTRUCTION ADDRESS	INSTR.	SUMMED INSTR. TIME BETWEEN ADDRESS	ORDINAL	JUMP ADDR	JUMP ORDINAL		PROC CALLED
UFSAD	MSV02231	HEAD1 002		00000000	001	00046	023		00255
UFSAD	MSV02231	00046	SCT	00000150	001				
UFSAD	MSV02231	00105Y	JLE	00008560	002	00113	003		
UFSAD	MSV02231	00112		00001110					
UFSAD	MSV02231	00113	LB	00000200	003			*	
UFSAD	MSV02231	00114 00130	DL	00000300	004			*	
UFSAD	MSV02231	00130Y	JGE	00003550	005	-00114	004		
UFSAD	MSV02231	00135Y	JNE	00000875	006	00143	008		
UFSAD	MSV02231	00140Y	JLT	00000550	007	00143	008		
UFSAD	MSV02231	00142		00000400					
UFSAD	MSV02231	00143	LA	00000150	008			*	
UFSAD	MSV02231	00157P	LBJ	00004080	009				CSQRT
UFSAD	MSV02231	00163Y	JGE	00000625	010	00166	012		
UFSAD	MSV02231	00165I	J	00000300	011	00202	018	X	
UFSAD	MSV02231	00166	LA	00000150	012			*	
UFSAD	MSV02231	00171Y	JGE	00000475	013	00173	014		
UFSAD	MSV02231	00172		00000250					
UFSAD	MSV02231	00173	LA	00000150	014			*	
UFSAD	MSV02231	00176Y	JG	00000475	015	00201	017		
UFSAD	MSV02231	00200Y	JLE	00000300	016	00202	018		
UFSAD	MSV02231	00201	BZ	00000250	017			*	
UFSAD	MSV02231	00202	LA	00000150	018			*	
UFSAD	MSV02231	00206		00001150					
UFSAD	MSV02231	00207 00223	DL	00000300	019			*	
UFSAD	MSV02231	00223Y	JGE	00003550	020	-00207	019		
UFSAD	MSV02231	00246Y	JNE	00005775	021	00253	022		
UFSAD	MSV02231	00252		00000600					
UFSAD	MSV02231	00253	BS	00000250	022			*	
UFSAD	MSV02231	00255E	J	00000350	023	00046	001	X	

exactly what a third program (the TIMING/DATA BASE QUERY PROGRAM) actually does to develop all the path times within a procedure, i.e., an enumeration technique.*

Since the number of paths in a procedure can be in the thousands, only summary statistics of the paths are presented; namely, the two smallest path times (Min 1 and Min 2), the maximum path time (Max), the average path time (AVG), the empirical and unbiased standard deviation of the path times (UN-STD EMP-STD), and the number of paths in each category (defined in the next paragraph).

Paths are grouped into the following catagories (or combinations of catagories): paths that contain loops, paths that do not contain loops, paths that do or do not contain procedures, executive state, or I/O calls. The category of paths containing calls and loops is further broken into subsets of paths that contain the same combination of calls, loops, and/or the same combination of calls contained within loops. Combinations are independent, i.e., exclusive (or in more general terms probabilistic). This makes it a simple matter to determine the exclusivity of calls and loops. Summary statistics are given for each catagory. The output of the third program (the TIMING PROGRAM) is given in Tables 4 and 5. (In Table 4 an asterisk beside a loop indicates it is an inner loop, i.e., a loop within a loop. In Table 5 loops are denoted by parenthesis pairs, i.e., ( ), and inner loops are denoted by parenthesis pairs with pairs. The address of the loop extremities is contained within the parenthesis. The loop single pass execution time statistics are given beside the corresponding loop addresses, and the statistics for the non-looped segments of these paths are in the "COMBO STATS" section.)

Figure 4 shows the steps of the methodology in terms of the automated tools and their inputs and outputs.

APPLICATION

The methodology of Chapter 2 works with a thread of procedures and a thread execution time. The Timing Program output on the other hand, is in terms of individual unlinked procedures and their multi-path execution times. It thus remains a task for the analyst to decide which procedures (and path in the procedure) will be selected to build the model thread. The best decision is based on knowledge of the existing and development systems/code (as delineated in the methodology in Chapter 2, i.e., pick procedures and paths within procedures that best characterize the system for which the model is to be built).

A careful analysis could even result in not just one path within a procedure being picked but statistically combining several paths and/or procedures (using weighting factors/path counts/procedure counts obtained from the Timing Program). On the other hand, if it is desired to do a quick/ inexpensive analysis, a worst case analysis could be done with minimum existing and development code knowledge by picking the maximum path/maximum number of procedure combinations from the Timing Program output.

*The algorithm used is the enumeration of a modified binary tree.

TABLE 4. OUTPUT SOFTWARE TIMING PROGRAM

PROC NAME	SYS-PROC	PAGE-NO	START ADDR	STOP ADDR	NUM PATHS		AVG TIME (USECS)	MIN1 TIME (USECS)	MIN2 TIME (USECS)	MAX TIME (USECS)	EMP-STD	UNB-STD
UFA0J5P8	MSV02231	35	00725	00767	1		71.55	71.55		71.55	0.00	
	CSVVPA	00734	00000									
	SY31											
UFAPOFF	MSV02231	34	00661	00724	1		107.35	107.35		107.35	0.00	
	CSVFLCS	00664	00000									
	CSSC	00672	00000		J FCLLCDIAG LBJ = NOFXIT							
UFCAP	MSV02231	23	00256	00660	0							
UFCAP	MSV02231	23			128		286.82	221.75	221.75	351.90	51.65	51.86
	UFAPOFF	00323	00661									
	CSORT	00605	00000									
	UFA0J5P8	00655	00725									
LOOP		•	00360	00374	128	1	38.50	38.50		38.50MN1P2MX	0.00	
LOOP			00335	00544	128	2	435.90	429.18		442.70MN1P2MX	6.80	9.61
	CSORT	01503	00000									
UFCMMC	MSV02230	213	05473	06303	0							
UFCMMC	MSV02230	213			16		436.88	322.50	322.50	557.10	113.33	117.05
LOOP			05535	05536	16	1	3.30	3.30		3.30MN1P2MX	0.00	
LOOP		•	05541	05565	16	1	36.05	36.05		36.05MN1P2MX	0.00	
LOOP			05540	05566	16	1	39.85	39.85		39.85MN1P2MX	0.00	
LOOP			05735	05756	16	1	3.30	3.30		3.30MN1P2MX	0.00	
LOOP		•	05762	05006	16	1	36.05	36.05		36.05MN1P2MX	0.00	
LOOP			05761	06007	16	1	39.85	39.85		39.85MN1P2MX	0.00	
LOOP			06064	06300	16	2	471.00	452.50		489.50MN1P2MX	18.50	26.16
	UFTALOG	06211	07727									
	UFSORT	06252	07715									
	UFSORT	06256	07715									
UFDMC	MSV02230	195	05150	05256	408		83.44	37.50	48.50	182.65	11.04	11.05
	UFDMOBT	05150	05257									
	SY1											
	SY1											
	CSRTC	05173	00000									
	SY1											
	*UFDMDFP	05203	00111									
	SY1											
	SY1											
	SY17											
	SY1											

TABLE 4. (CONT.)

PROC NAME	SYS-PROC	PAGE-NO	START ADDR	STOP ADDR	NUM PATHS		AVG TIME (USECS)	MIN1 TIME (USECS)	MIN2 TIME (USECS)	MAX TIME (USECS)	EMP-STD	UNB-STD
UFDMD	MSV02230	195			204		110.69	68.50	71.50	127.65	10.65	10.67
	UFDMDBT	05160	05257									
	SY1											
	SY1											
	CSRTC	05173	00000									
	SY1											
	*UFDMDPR	05203	00111									
	SY1											
	SY1											
	SY17											
	SY1											
LOOP			05224	05233	204	1	13.75	13.75		13.75M1P2MX	0.00	
UFEXTRAP	MSV02230	247	07155	07262	1		227.00	227.00		227.00	0.00	
UFGENLO	MSV02230	235	06540	07121	0							
UFGENLO	MSV02230	235			432		376.06	347.30	347.30	388.10	6.78	6.79
	UFSORT	06652	07715									
	UFSORT	06665	07715									
	UFSORT	07001	07715									
	UFSORT	07013	07715									
LOOP			06721	06753	432	2	130.35	129.80		130.90M1P2MX	0.54	0.78
LOOP			06720	06754	432	2	134.15	133.60		134.70M1P2MX	0.54	0.78
LOOP			07046	07066	432	2	76.60	76.05		77.15M1P2MX	0.54	0.77
LOOP			07045	07067	432	2	80.40	79.85		80.95M1P2MX	0.54	0.77
UFLOPACK	MSV02230	249	07263	07364	0							
UFLOPACK	MSV02230	249			1024		83.00	73.00	73.00	93.00	3.53	3.53
LOOP			07265	07277	1024	4	13.80	10.80	13.80	16.80M1P2MX	2.12	2.44
UFPACK	MSV02230	245	07122	07154	5		13.50	10.50	13.50	18.30	2.84	3.18
	UFLOPACK	07136	07263									
UFPACK	MSV02230	245			4		30.05	27.80	29.30	32.30	1.67	1.93
	UFLOPACK	07136	07263									
LOOP			07140	07145	4	1	10.35	10.35		10.35M1 MX	0.00	
	UFEXTRAP	07140	07155									
	UFLOPACK	07141	07263									

TABLE 4. (CONT.)

PROC NAME	SYS-PROC	PAGE-NO	START ADDR	STOP ADDR	NUM PATHS		AVG TIME (USECS)	MIN1 TIME (USECS)	MIN2 TIME (USECS)	MAX TIME (USECS)	EMP-STD	UNB-STD	D
***UFSD	MSVDJ223J	265	07756	10234	13000		114.33	48.25	49.75	148.15	15.58	15.58	
	CSALERTQ 13000	00000		J FOLLOWING LBJ = NOEXIT									
UFDS4	MSV02230	262	16427	16464	19		32.71	19.58	28.58	48.18	8.94	9.28	
UFSAD	MSV02231	1A	00046	00255	0								
UFSAD	MSV02231	1A			84		252.85	226.15	231.65	278.25	9.12	9.17	
	CSOPT	03157	00000										
LOOP			00114	00138	84	1	38.58	38.58		38.58M1P2MX	0.00		
LOOP			00207	00223	84	1	38.58	38.58		38.58M1P2MX	0.00		
UFSOPT	MSV02230	262	07715	07726	1		39.88	39.88		39.88	0.00		
UFSTAB	MSV02230	232	06442	06537	2		198.38	197.48		199.28	0.90	1.27	
	UFCMD	06445	06304										
	UFTALOG	06453	07727										
	UFTALOG	06457	07727										
	UFSOPT	06512	07715		J FOLLOWING LBJ = 2								

TABLE 5. OUTPUT OF SOFTWARE TIMING PROGRAM

```
********** PROC NAME NAPIM      00645 01016 **********
```

	# OF PATHS	AVG TIME	MIN 1 TIME	MIN 2 TIME	MAX TIME
PATHS WITH NO LOOPS-NO PROC CALLS00645 01016	3	19.50	6.00	17.00	35.50
PATHS WITH LOOPS - NO PROC CALLS	1	57.00	57.00		57.00

PATHS WITH PROC CALLS (AND LOOPS):

PATHS COMBINATION 1

(00667 (00671 00676) (00704 00707) 00713) (00717 00722) (00725 00727) (00733 CSLLXY
CSLLXY CSLINE 01015)

COMBO STATS	1	54.00	54.00	54.00	54.00

LOOP	ADDRESS				
00671	00676	10.00	10.00	10.00	10.00
00704	00707	6.00	6.00	6.00	6.00
00667	00713	51.00	51.00	51.00	51.00
00717	00722	6.50	6.50	6.50	6.50
00725	00727	4.50	4.50	4.50	4.50
00733	01015	75.40	75.40	75.40	75.40

PATHS COMBINATION 2

(00667 (00671 00676) (00704 00707) 00713) (00717 00722) (00725 00727) (00733 CSLLXY
CSLINE 01015)

COMBO STATS	1	54.00	54.00	54.00	54.00

00671	00676	10.00	10.00	10.00	10.00
00704	00707	6.00	6.00	6.00	6.00
00667	00713	51.00	51.00	51.00	51.00
00717	00722	6.50	6.50	6.50	6.50
00725	00727	4.50	4.50	4.50	4.50
00733	01015	70.10	70.10	70.10	70.10

PATHS COMBINATION 3

(00733 CSLLXY CSLLXY CSLINE 01015)

COMBO STATS	2	23.25	14.00	14.00	32.50
00733 01015		75.40	75.40	75.40	75.40

PATHS COMBINATION 4

(00733 CSLLXY CSLINE 01015)

COMBO STATS	2	23.25	14.00	14.00	32.50
00733 01015		70.10	70.10	70.10	70.10

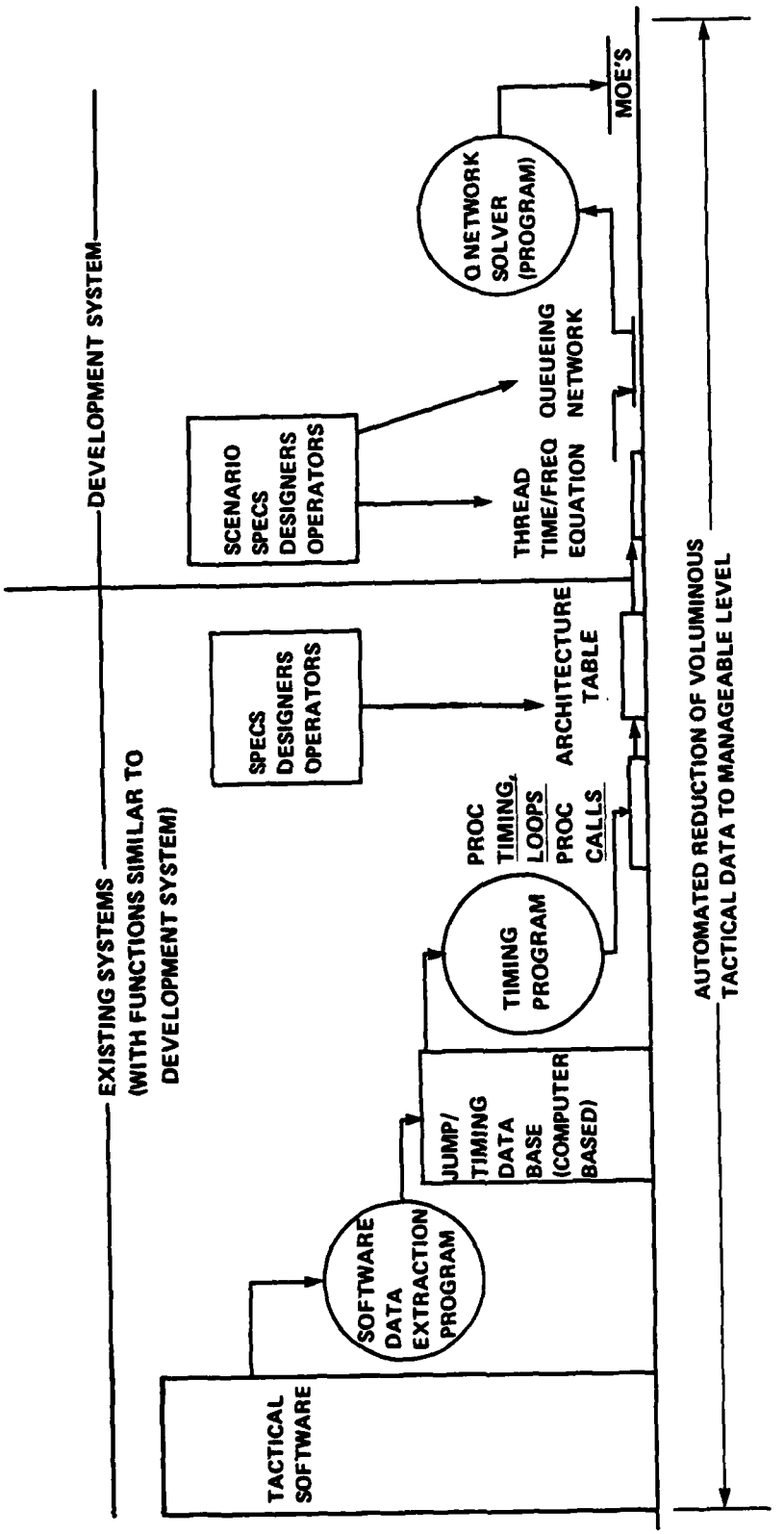


FIGURE 4. THE METHODOLOGY'S STEPS AND TOOLS

The Timing Program is designed to either aid in picking a path and/or provide information about a picked path. Consider the following examples:

1. The Min 2 path is much larger than the Min 1 path: this usually indicates the Min 1 path is just an error path and probably is not executed as a realistic minimum path, i.e., the assumption is usually made that the code executes without errors. Thus the Min 1 path should be excluded and an AVG, Max, or some intermediate path should be picked.

2. Min 1 or Min 2 time is very close to the Max time or the standard deviation is small: in this case all path times are approximately equal and any path can be used with confidence.

3. AVG and Max path times are close but far from Min 1 and Min 2 times: this indicates either a very large max path or a large number of paths above average. Which case applies can be checked by looking at the "number of paths" in the timing program statistics.

The grouping of paths into combinations allows the analyst to gain a quick understanding of a procedure or even indicate where in a procedure the analyst should look to make a decision. For example in Table 5 the timing program output shows NAPIM has two modes of evaluation; jumping around the initial loops to the loop at 733 (combinations 3 and 4); or doing all the loops (combinations 1 and 2). Also it is seen that the second call to CSLLXY is the only call that is conditional. i.e., the other procedure calls occur in every combination and thus occur in every path in NAPIM. In order for the analyst to decide which paths to choose he need only look at the jump condition around the initial loops, (seen to be between the beginning of the program, location 645, and the first loop, location 667), and the condition controling the second call to CSLLXY, (seen to be between 733 and 1015).

If the analyst needs more information than is provided in the timing program's normal output, he can select an option to output all the paths/path times within a procedure.

As an aid to developing thread times two other programs can be used. One program outputs all the procedures of the "existing code" formatted into threads expressed as an architecture table and topology as shown in Table 1 and Figure 1. (See Chapter 2 for an explanation of Table 1 and its threads.) This architecture program also outputs the procedure names that could be used to make up the threads. This list of threaded procedure names can be used as input to the Timing Program, i.e., it automatically tells the timing program which procedures to select and time from the data base. (Of course this list can be edited to form the thread that best models the system being investigated.)

The second program (still in development) is an interactive program to sum individual procedure times into thread times, i.e., it develops the "thread time equation."

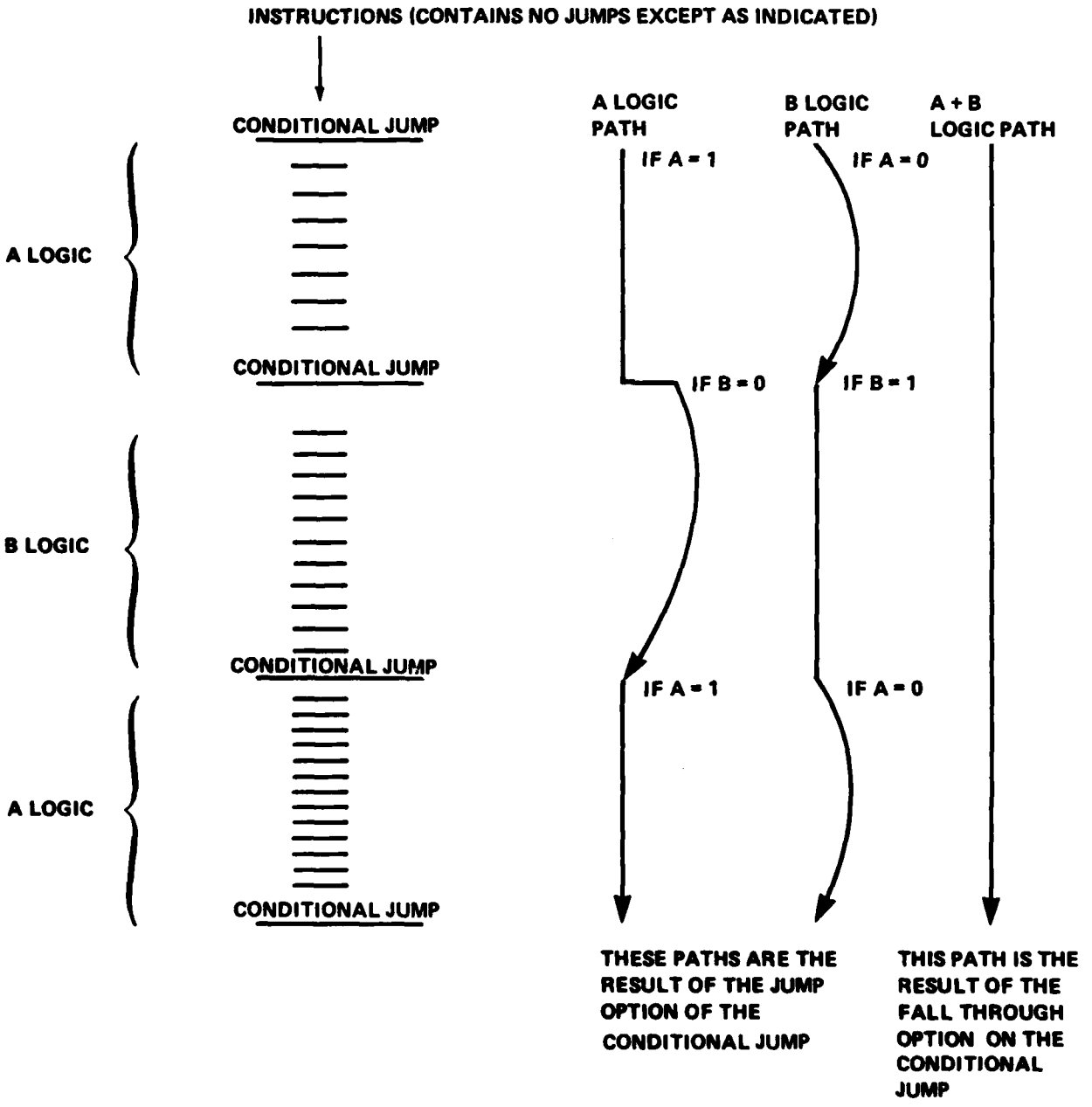
ASSUMPTIONS AND RAMIFICATIONS

As has been described the Timing Program uses the technique of enumeration to develop procedure times. The Timing Program assumes every path is

independent and equally probable. In reality this assumption is not true because path probabilities will depend on data values and/or data probabilites evaluated at the jump points. As described in the last section the violation of this assumption is corrected by forcing the analyst to gain knowledge of the codes being analyzed (by utilizing the methodology's tool).

There is one software architecture that the analyst should be aware of where the Timing Program will time paths that have low or even possibly zero probability of occurrence. These low probability paths tend to skew the timing statistics to larger values. The situation occurs when a code has two variables that are mutually exclusive (e.g., when A=1, B=0 and when B=1, A=0), the code logic for Case A and Case B is interspersed within the same proc, and the logic segments are separated by conditional jumps. For this software architecture the Timing Program will generate a path that is the sum of Case A and Case B logic (i.e., a path where A=B), which is the maximum path since it is the sum of both A and B path logics (see Figure 5). This maximum (sum of A and B) path is never supposed to be executed because it violates the data logic about the variables A and B. However, because the architecture shown in Figure 5 has conditional rather than unconditional jumps, the logic of the architecture (as developed by the compiler) does not preclude the path where A=B. In order for the programmer to insure the execution logic matches the variable logic he must test for the Case A=B as a separate conditional jump. If he does not do this and just relies on the believed inviolate properties of the variables, then it must be assumed that the execution logic and architecture logic is identical, i.e., the Case A=B can occur and it is truly the maximum path.

In order for the analyst to detect this particular architecture the timing program prints out the jump addresses of the max path. (It should be noted that if the Max path is incorrect then the average path time and the number of paths will also be incorrect.)



(CONDITIONAL JUMP = A jump that can either fall through to the next instruction or jump to some other instruction depending on the result of the condition tested.)

FIGURE 5. EXAMPLE OF AN ARCHTECTURE THAT THE TIMING PROGRAM MAY GIVE INVALID RESULTS

CHAPTER 4

VALIDITY AND APPLICATIONS

VALIDITY/CASE STUDIES

The automated tools have been extensively checked against manual code timing calculations with near perfect agreement.

Reference 2 showed that the accuracy for a case where the methodology was applied to a development architecture similar to an existing architecture was 6 percent. For this case the existing system's execution time was measured with a software debug utility and the development system was measured with the methodology. The existing system was the Mk 116 Mod 4 TMA Function and the development system was the ASWCS Model 1 TMA Function.

Reference 3 modeled the AN/UYK-25 SIMAS Acoustic Prediction Performance Calculators execution times and compared results to manual stop watch measurements. The results were 19 versus 20.5 seconds (7 percent) for the DIMUS case and for the ACTIVE Case 44 versus 53 seconds (20 percent).

Reference 4 used a hardware monitor to measure Navy tactical software execution times and then built POD queueing models. A comparison of the measurements indicated agreement to within 6 to 12 percent accuracy. This result is essentially an accuracy measurement on the use of POD/queueing theory as a computer modeling technique for Navy systems.

Volume II examines in detail the application of the methodology to a test case of a dissimilar architecture between the development system and the existing system. For this test case the Mk 116 Mod 1 ASW Fire Control System's (FCS) Weapon Control Function (WCF) was assigned to be the development system and the Mk 116 Mod 0 FCS as the existing systems. A model of the Mod 1 WCF was constructed using the functions specified in the Mod 1 PPS/PDS (Reference 5), and Mod 0 based execution times were developed for these Mod 1 functions. The model results were then compared to the actual Mod 1 execution times. Both Mod 1 and Mod 0 execution times were developed using the methodology's automated tools. The accuracies range from $\pm$7 percent to $\pm$25 percent for the first level (tier 1) subfunctions that make up the Mod 1 WCF. (It should be noted that one WCF procedure could have been modeled in two ways and one way gave an error of 50 percent. See Volume II for details.)

Using the frequency of each of the Mod 1 WCF procedures (given in Table 1 of Volume II) and the errors calculated for each procedure (from Volume II) a weighted average error can be calculated for the entire WCF function. The result is:

±13 percent (not using the 50 percent error in the average)

±21 percent (using the 50 percent error in the average)

APPLICATIONS

The methodology has been successfully applied to the following Navy systems:

Mk 116 Mod 0 FCS
Mk 116 Mod 1 FCS
AYQ/25 - SIMAS Acoustic Prediction System
ASWCS Model 1
FFG-7 WAP/LAMPS
Mk 116 Mods 5,6,7 (Systems in development, no code exists)
PDP FORTRAN (Prototype Mk 116 Mod 7 Correlation Software)

Results and details of applying the methodology are documented in References 2 and 3 and in Volume II. Volume II, in point of fact, besides showing accuracy, illustrates in detail that if necessary, the methodology can be applied to the fine level of just groups of instructions within individual procedures.*

*The timing program has an option to allow it to time code in an inputted address range (within a procedure) instead of an entire procedure.

CHAPTER 5

CONCLUSIONS

VALIDITY

There are almost no statements of absolute mathematical certainty that can be made concerning models, modeling, and thus modeling methodologies applied to systems in development. The reason for this is that a model's system development utility is in its predictive ability. The only way to be certain that the model prediction is truly accurate is to compare it to measurements on the developed system. However, if you have the developed system to measure, and you can indeed measure it, you do not need the model to predict anything.* Because of this circular problem a system development model can never be validated until such time that it is not worth validating.

There is one exception to this worthlessness of validating a model and that is to gain some data concerning the accuracy of the modeling technique/methodology. Note the term used was "gain some data" versus "prove correct." Validation of a model is usually only a case study, not a mathematical proof of correctness. Even after doing a large number of case studies one has not proved there exist no other case that gives a different result from the previous cases.

The validity results of this report (as well as 100 percent of all the results concerning predicative modeling accuracy) fall into the area of limited validity that is true of case studies. With this clearly understood the following can be said:

1. The methodology presents a viable/useable approach to assess computer system performance.

2. The methodology can give results that are within acceptable limits of accuracy.

APPLICATION ENVIRONMENT

In order to apply this methodology and get good results a certain organizational environment is necessary. The necessary environment is one where there is a free flow of information between the assessment analyst and the architects/designers/programmers (i.e., the developers) of the system being assessed.

*This is not true for the cases of system tuning, non-major upgrading, and varying numerous parameters/scenarios as applied to an existing system.

It makes no sense for the analyst to be building a model based on outdated information, unless nothing else is available. While the above statement is immediately obvious, in most system development environments it is usually the opposite that is true, i.e., the analyst is using old information or he is waiting for a development milestone to be met so he can get information which by the time he gets it is old, i.e., he is always behind the developer rather than current with him. Another frequent situation is that the developer says he cannot give out the information because he has not chosen one of several alternative approaches yet.

In all the above situations it would be more cost effective to give the analyst information as early as possible. If this is done the analyst could use the methodology to resolve design alternatives rather than come out with less meaningful results because he is using unreliable data.

Another reason for an open environment is that the accuracy of the methodology (as stated in Chapters 2 and 4) is directly dependent on knowledge of the software being analyzed. The tools of the methodology allow an analyst with minimal knowledge of the software being analyzed to complete the model. However, the time/money needed to complete the model to a good accuracy is greatly reduced if the proper expertise from the developers is available.

Since the methodology specifies what type of information/data is needed to carry out an assessment (e.g., loop and branch data), the most efficient information exchange between developer and analyst would be for the developer to incorporate the required data directly into the software in a machine readable form. Appendix A gives a scheme for incorporating the required data using machine readable embedded comments. Other schemes such as formatting of the executable code or a type of data dictionary is also possible.

The tools and methodology are most efficiently employed on non-convoluted architectures that are made up of short non-convoluted/simple procedures. Since simplicity and short procedures are also the basic tenants of modern software design (e.g., structured programming, etc.), there should be no problem to require software be written in a form that is efficient for assessment by the methodology.

APPLICATIONS

This methodology and its tools have evolved over a period of 3 years. The tools have gone through many iterations to make them more comprehensive and more efficient. These tools have been applied to a variety of Navy tactical software and in one instance FORTRAN prototype software. Even though the methodology is not totally automated, NSWC's use of the methodology tools has shown that a 2 Bay AN/UYK-7 plus disk based system can be modeled quickly enough to keep pace with the development process by two analysts that have no previous knowledge of any software being analyzed.

SYNTHESIS AND ANALYSIS METHODOLOGIES

A synthesis technique is one which is applied to a set of input variables/requirements, and it outputs an architecture/design that meets the requirements. An analysis technique is one which is applied to an architecture/design, and it outputs an analysis of the architecture/design. The analysis shows the weak points of the design. The design (weak points) can be modified/improved via intuition/expertise, and the design is re-input to the analysis technique. Thus by iterating the analysis technique the design can usually be made to meet its requirements. By this method an iterated analysis technique approaches a synthesis technique.

Thus the methodology of this report is not just an analytic performance assessment methodology but indeed a complete development/synthesis methodology.

CONCLUSION

It is the conclusion of this report that the methodology represents an acceptable approach for both the performance assessment and the architecture/design documentation of existing systems, and the development of new computer-based systems.*

*Assumes methodology presented would be combined with existing methods to include techniques for the generation of requirements, functional analysis, software generation, and testing.

CHAPTER 6

COMPUTER SYSTEM ASSESSMENT VIA THE METHODOLOGY VERSUS DIRECT MEASUREMENT

The obvious question that most development organizations ask is that since I have, or will have, the actual system, rather than apply an analytic technique, that can never be 100 percent validated, wouldn't it be cheaper, quicker, and more accurate just to measure the system? The not so obvious answer is that in practice it may not be cheaper, quicker, or for that matter possibly even more accurate.

In order to measure code you must first analyze the code to find the addresses of points of interest to measure. Most hardware monitors only allow a limited number of probes to be set, which means that all the measurements cannot be made at one time. If you are using a software monitor or DEBUG UTILITY you cannot use too many probes because of the possibility of slowing the running time of the system. In order to make the measurement, the code must be executing which means a software driver needs to be written. All of these factors are not trivial items in terms of time and money.

Once the measurement has been made, what has really been measured, the system or the software driver/system combination? The answer is, of course, the combination, because if the driver is changed the measurement will change. That means in order to understand the meaningfulness of any measurement, the code must be analyzed to understand exactly how the driver is driving it. But this driver/code analysis is tracing the effect of code variables, set by the driver. But this is the same process that was called "threading" in the methodology.*,** Unless this driver/code analysis is done, there is no way to insure what timing path the driver is executing the code on, i.e., a maximum time path, a minimum time path, or some intermediate time path. The methodology, on the other hand, forces the analyst to explicitly know the execution path.

A high-quality hardware monitor was used to analyze a Navy system equivalent in size to a 2 Bay AN/UYK-7 with disk.[4] It took on the order of 1 man-year of code knowledgeable personnel to completely analyze the system. This is probably about the same amount of time it would take equivalent personnel to use the methodology.

*See steps 4 and 6 in Chapter 2, and the application section of Chapter 3.

**In actuality the designing of the driver is equivalent to the methodology's threading.

How is a hardware or software monitor validated? They are usually validated by counting and summing pulses, cycles, or instructions on as many system/code samples that is practical. But this is exactly the same validation technique used on the methodology's tools. Thus the level of validation of both measurement and the methodology is theoretically the same and in practice depends on the degree of thoroughness of the validation process.

The bottom line of measurement versus methodology is that a thorough analysis and a well done measurement are probably roughly equivalent in terms of time, money,* and accuracy.

The advantages of a validated model over direct measurement are:

1. It is cheaper, easier, and quicker to investigate the effect of parameterizations and various scenarios, i.e., perform trade-off studies or discover catastrophic situations.

2. System timing can be easily investigated.

3. Limited system prediction/upgrade can be easily investigated.

4. Expensive measurement equipment does not need to be purchased.

5. Measurement systems usually do not calculate Queueing MOEs (see Table 2); whereas models usually do.

The main advantage of modeling over direct measurement comes early in the design process when there is nothing to make measurement on. At this stage, modeling can help direct the design and prevent building a system that can have catastrophic failure. On the other hand, one way to validate a model is with direct measurement driven by a well documented driver.

The ideal situation would be to have both measurement and methodology available because they certainly could be used to complement each other at different stages of the development process.

*If the system already has a driver (or "wraparound simulator") available, and in its development it was documented as to what code thread/path/addresses it was driving the code on, then direct measurement might be quicker and cheaper than the methodology.[2] However, usually it is not known how the driver is driving the code. Using results from unknown driver paths is doing the blackest of black box analysis.

REFERENCES

1. Buzen, Performance Oriented Design (POD) Reference Manual, BGS Systems, Waltham Massachusetts, Sep 1980.

2. Franklin, J., Models for the Architecture of the ASW Control Systems (ASWCS) Mk 116 Mods 5, 6, and 7, NSWC TR 83-338, in publication.

3. Gray, C., Computer Modeling and Architecture of the AN/UYK-25 SIMAS, NSWC TR 83-48, Feb 1983.

4. Cooke, J. Performance Modeling of the Trident Fire Control System, NSWC TR 81-350, Mar 1982.

5. Computer Program Performance and Design Specification for Underwater Fire Control System Mk 116 Mod 1, WS14521, 1 May 1976, Naval Sea Systems Command.

APPENDIX A

SOFTWARE FORMAT STANDARDS FOR MAXIMIZING THE AUTOMATED CAPABILITIES OF THE CSE TOOLS

In order to maximize the automated capabilities of the tool, the following software format standards can be implemented. Code written to these stardards could be scanned by another program to extract the information needed by the methodology.

1. All comments be delineated with a fixed non-alphanumeric character.

2. Before each procedure an explanatory sentence be given. Each line of the sentence should start and stop in a fixed column.

3. Major branch points within a procedure be commented with explanation. Comments start and stop in fixed columns.

4. Same as 3 but applied to every procedure call instead of branch points.

5. Same as 3 but applied to every loop instead of branch points.

6. Loop comments include the minimum, maximum, and average values the loop variables can assume. Also loop variables are explained.

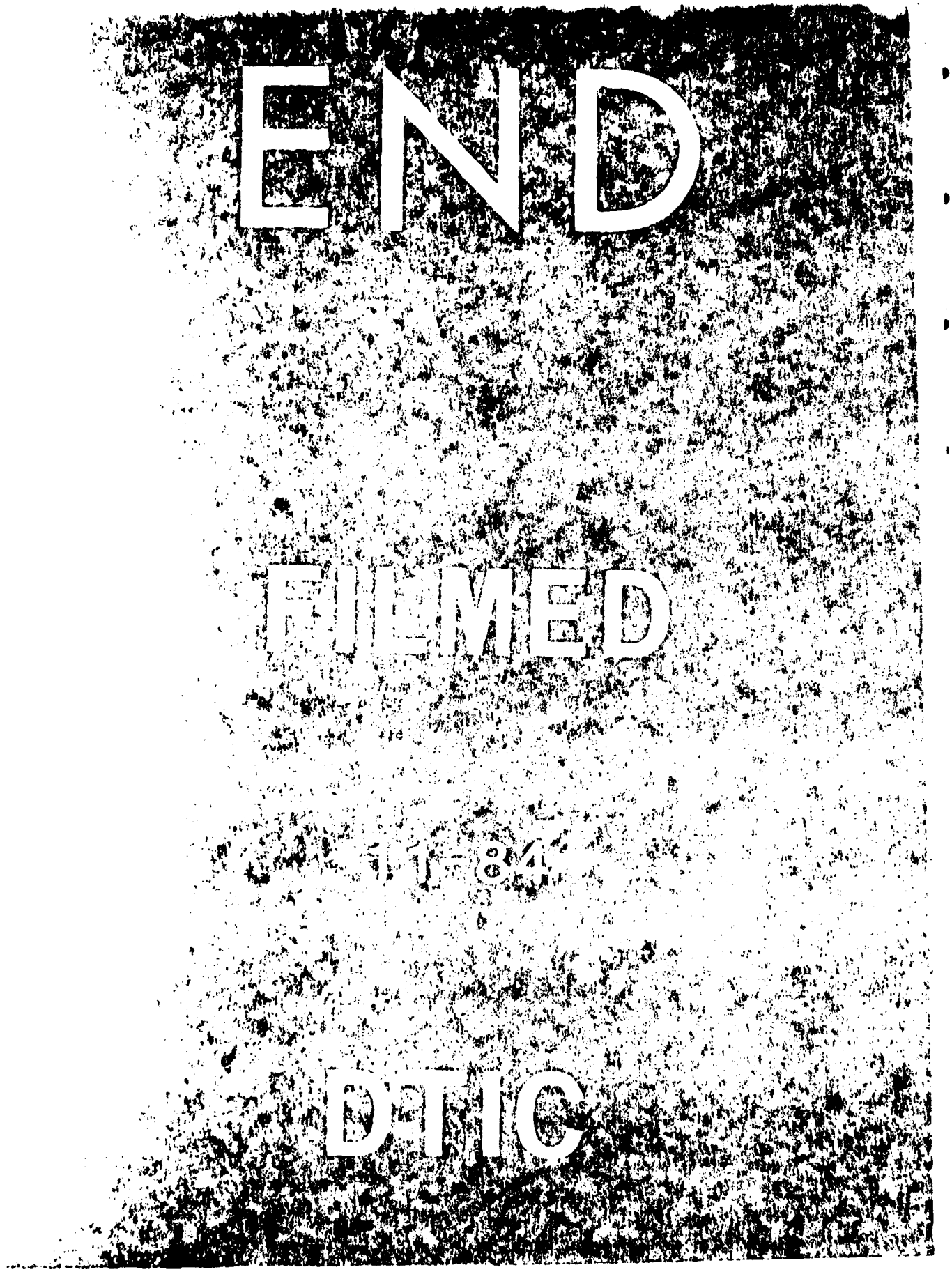
7. Loop values (of 6) and loop variables be delineated by fixed non-alphanumeric characters in fixed columns.

DISTRIBUTION

	Copies
Commander Naval Underwater Systems Center Attn: Code 33B (R. Prager) Code 33B (B. Thorpe) New London, CT 06320	 1 1
Commander Naval Underwater Systems Center Attn: Code 3531 (P. Nadeau) Bldg 1171-2 Newport, RI 02841	 1
Commander Naval Ocean Systems Center Attn: Code 82 (Dr. R. Kolb) Code 6201 (J. Reardon) Code 62 (R. Thulen) Code 6211 (M. Stonebreaker) Code 6212 (L. Fransdal) Code 6201 (D. Callabro) Code 8324 (Sutton) San Diego, CA 92152	 1 1 1 1 1 1 1
Commander Naval Sea Systems Command Attn: SEA-61V (R. Wilson) SEA-61V (D. Perrill) SEA-61R (M. Wapner) PMS-400 (R. Hill) PMS-400 (CAPT Donegan) SEA-61R2 (P. Andrews) PMS-400 PMS-409 PMS-411 PMS-411B PMS-411C PMS-411G PMS-411E Washington, DC 20362	 1 1 1 1 1 1 1 1 1 1 1 1 1
Commander Naval Supply Systems Command Attn: Code 0339 (G. Bernstein) Washington, DC 20376	 1
Commander Naval Electronic Systems Command Attn: ELEX-G134 (J. Machado) Washington, DC 20363	 1
NASA Langley Research Center Attn: Ed Dean/Mail Stop 444 Hampton, VA 23665	 1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	 12
General Electric Corporation Attn: Andy Rasi Bldg. 1, Rm. R5 Farrell Road Plant Syracuse, NY 13221	 1
BGS Systems, Inc Attn: Jeff Buzen 470 Totten Pond Rd Waltham, MA 02254	 1
Sperry (Systems Management) Attn: J. E. Zellers Great Neck, NY 11021	 1
Library of Congress Attn: Gift and Exchange Division Washington, DC 20540	 4

DISTRIBUTION (Cont.)

	Copies
Sperry (Computer Systems)	
Defense Syst. Div	
Attn: S. C. Andersen	1
J. W. Albers	1
3333 Pilot Knob Rd	
St. Paul, MN 55122	
RCA	
127-331 Marine Hwy	
Attn: Lee Fleisher	1
Morristown, NJ 08057	
Federal Computer Performance Evaluation and Simulation Center	
Attn: D. Ball	1
J. Wethersbee	1
6118 Franconia Rd.	
Alexandria, VA 22310	
University of Maryland	
Attn: Dr. S. Tripathi	1
Rm. 4333	
Computer and Space Sciences Dept.	
College Park, MD 20742	
Internal Distribution:	
E34	1
E431	9
E432	3
G	1
K34	1
K34 (A. Wrenn)	1
K54	1
K54 (W. McCoy)	1
K54 (J. Cooke)	1
N	1
N04 (Dr. H. Crisp)	1
N05 (C. Yarbrough)	1
N10	1
N13 (D. Mensch)	1
N14 (W. Martin)	1
N20	1
N21 (M. Masters)	1
N21 (D. McConnell)	1
N30	1
N32 (N. Harder)	1
N51 (E. Price)	1
N53 (D. C. Hill)	1
N302 (R. Cullen)	1
N305	1
N307 (J. Straub)	1
U	1
U04	1
U05	1
U05 (B. Podolsky)	1
U20	1
U22	1
U23 (J. Cottrell)	1
U30	1
U31	1
U31 (H. Ng)	1
U31 (J. Simpson)	1
U32	1
U32 (J. Franklin)	3
U32 (C. Gray)	1
U32 (H. Herring)	1
U32 (R. D. Timberlake)	1
U04 (M. Stripling)	1


END
FILMED
11-84
DTIC